Document number: D0631R1
Date: 2017-08-03
Project: Programming Language C++
Audience: Library Evolution Working Group, SG6 (Numerics)
Reply-to: Lev Minkovsky lminkovsky@outlook.com

Math Constants

1. Changelog

Changes from R0:

- Added changelog, header and footer
- Several readability improvements
- Chapters are numbered
- The fourth chapter subdivided into subchapters
- Design goals are stated
- A different set of constants is proposed
- Naming conventions are different
- A drop-in replacement for POSIX constants is no longer proposed
- All definitions are in the new math_constants namespace
- Variable template types are inline
- Added new implementation requirements
- *float* and *double* typed constants are proposed
- Usage scenarios added
- The 5th and 6th chapters are reworked according to the abovementioned changes
- A link to the list of built-in Wolfram constants added to the 7th chapter

2. Introduction

C++ inherited from C a rich library of mathematical functions, which continues to grow with every release. Amid all this abundance, there is a strange gap: none of the major mathematical constants is defined in the standard. This proposal is aimed to rectify this omission.

3. Motivation

Mathematical constants such as π and *e* frequently appear in mathematical algorithms. A software engineer can easily define them, but from their perspective, this is akin to making a reservation at a restaurant and being asked to bring their own salt. The C++ implementers appreciate this need and attempt to fulfil it with non-standard extensions.

The IEEE Standard 1003.1™-2008 a.k.a POSIX.1-2008 stipulates that on all systems supporting the X/Open System Interface Extension, "the <math.h> header shall define the following symbolic constants. The values shall have type *double* and shall be accurate to at least the precision of the *double* type."

`M_E` - value of e
`M_LOG2E` - value of $\log_2 e$
`M_LOG10E` - value of $\log_{10} e$
`M_LN2` - value of ln2
`M_LN10` - value of ln10
`M_PI` - value of π
`M_PI_2` - value of $\frac{\pi}{2}$
`M_PI_4` - value of $\frac{\pi}{4}$
`M_1_PI` - value of $\frac{1}{\pi}$
`M_2_PI` - value of $\frac{2}{\pi}$
`M_2_SQRTPI` - value of $\frac{2}{\sqrt{\pi}}$
`M_SQRT2` - value of $\sqrt{2}$
`M_SQRT1_2` value of $\frac{\sqrt{2}}{2}$

POSIX.1-2008 explicitly states that these constants are outside of the ISO C standard and should be hidden behind an appropriate feature test macro. On some POSIX-compliant systems, this macro is defined as _USE_MATH_DEFINES, which led to a common assumption that defining this macro prior to the inclusion of math.h makes these constants accessible. In reality, this is true only in the following scenario:

1) The implementation defines these constants, and
2) It uses _USE_MATH_DEFINES as a feature test macro, and
3) This macro is defined prior to the first inclusion of math.h or any header file that directly or indirectly includes math.h.

These makes the availability of these constants extremely fragile when the code base is ported from one implementation to another or to a newer version of the same implementation. In fact, something as benign as including a new header file may cause them to disappear.

The OpenCL standard by the Kronos Group offers the same set of preprocessor macros in three variants: with a suffix _H, with a suffix _F and without a suffix, to be used in fp16, fp32 and fp64 calculations respectively. The first and the last sets are macro-protected. It also defines in the cl namespace the following variable templates:

e_v, log2e_v, log10e_v, ln2_v, ln10_v , pi_v, pi_2_v, pi_4_v, one_pi_v, two_pi_v, two_sqrtpi_v, sqrt2_v, sqrt1_2_v,

as well as their instantiations based on a variety of floating-point types and abovementioned macros. An OpenCL developer can therefore utilize a value of cl::pi_v<*float*>; they can also access cl::pi_v<*double*>, but only if the cl_khr_fp64 macro is defined.

The GNU C++ library offers an alternative approach. It includes an implementation-specific file ext\cmath that defines in the __gnu_cxx namespace the templated definitions of the following constants:

__pi,__pi_half,__pi_third,__pi_quarter,__root_pi_div_2,__one_div_pi,__two_div_pi,__two_div_root_pi ,__e,__one_div_e, __log2_e, __log10_e, __ln_2, __ln_3, __ln_10, __gamma_e, __phi, __root_2, __root_3,__root_5, __root_7, __one_div_root_2

The access to these constants is quite awkward. For example, to use a *double* value of π, a programmer would have to write __gnu_cxx::__math_constants::__pi<*double*>.

All these efforts, although helpful, clearly indicate the need for standard C++ to provide a set of math constants that would be both easy to use and appropriately accurate.

4. Design Considerations and Proposed Definitions

4.0. Design goals.

1) The user should be able to easily replace all POSIX constants with standard C++ constants.
2) The constants should be available for all fundamental floating-point types without type conversion, and should be defined with maximum precision of their respective types.
3) It should be possible to easily create a set of values of basic trigonometric functions of common angles, also with their maximum precision.
4) The standard constants should include the commonly used predefined constants of the Mathematica's Wolfram language.
5) They should be at least as concise and readable as POSIX constants.
6) They shouldn't cause name collisions and build breakages.
7) It should be possible to instantiate them for user defined types.

4.1. Which constants to standardize?

To achieve the design goals 1), 3) and 4) we need to provide the following constants:

e	- value of e
log2e	- value of $\log_2 e$
log10e	- value of $\log_{10} e$
ln2	- value of ln2
ln10	- value of ln10
pi	- value of π
invpi	- value of $\frac{1}{\pi}$
invsqrtpi	- value of $\frac{1}{\sqrt{\pi}}$
sqrt2	- value of $\sqrt{2}$
sqrt3	- value of $\sqrt{3}$
invsqrt3	- value of $\frac{1}{\sqrt{3}}$
radian	- value of $\frac{180}{\pi}$
egamma	- value of Euler-Mascheroni γ constant
phi	- value of golden ratio constant ϕ = (1+$\sqrt{5}$)/2

catalan - value of Catalan's constant
apery - value of Apéry's $\zeta(3)$ constant
glaisher - value of Glaisher's constant

The alternative naming of inverse constants could be inv_pi, inv_sqrtpi and inv_sqrt3. Any underscore usage beyond this would jeopardize the design goal 5).

It should be noted that all fundamental floating-point types are stored internally as a combination of a sign bit, a binary exponent and a binary normalized significand. If a ratio of two floating-point numbers of the same type is an exact power of 2 (within a certain limit), their significands will be identical. Therefore, in order to achieve the design goal 1), we don't have to provide replacements for both M_PI and M_PI_2 and M_PI_4. The user will be able to divide the M_PI replacement by 2 and by 4 and achieve goals 2), 3) and 5).

4.2. Where should they be defined?

As stated in the design goal 6), it is essential to avoid possible name collisions with the existing customer code base. For example, if we are to introduce an std::e constant, every source file with both "using namespace std" and a user variable "e" would no longer compile. The GNU C++ library resolves this problem by using the __math_constants namespace. The C++ standard already has an std::regex_constants namespace, presumably serving the same purpose. There appears to be a strong existing precedent for an introduction of a new namespace std::math_constants. Without it, we would have to make variable names long enough to minimize the chance of collisions. This would not help our efforts to achieve the design goal 5).

4.3. How should they be defined?

Math constant definitions should begin with the following set of templates:

```
template<typename T > inline constexpr T e_t
template<typename T > inline constexpr T log2e_t
template<typename T > inline constexpr T log10e_t
template<typename T > inline constexpr T pi_t
template<typename T > inline constexpr T invpi_t
template<typename T > inline constexpr T invsqrtpi_t
template<typename T > inline constexpr T ln2_t
template<typename T > inline constexpr T ln10_t
template<typename T > inline constexpr T sqrt2_t
template<typename T > inline constexpr T sqrt3_t
template<typename T > inline constexpr T invsqrt3_t
template<typename T > inline constexpr T radian_t
template<typename T > inline constexpr T egamma_t
template<typename T > inline constexpr T phi_t
template<typename T > inline constexpr T catalan_t
template<typename T > inline constexpr T apery_t
template<typename T > inline constexpr T glaisher_t
```

Alternatively, we can place template definitions into their own namespace:

```
namespace std {
	namespace math_constants {
		namespace templates {
			template<typename T > inline constexpr T e
```

```
                        template<typename T > inline constexpr T log2e
                        template<typename T > inline constexpr T log10e
                        template<typename T > inline constexpr T pi
                        template<typename T > inline constexpr T invpi
                        template<typename T > inline constexpr T invsqrtpi
                        template<typename T > inline constexpr T ln2
                        template<typename T > inline constexpr T ln10
                        template<typename T > inline constexpr T sqrt2
                        template<typename T > inline constexpr T sqrt3
                        template<typename T > inline constexpr T invsqrt3
                        template<typename T > inline constexpr T radian
                        template<typename T > inline constexpr T egamma
                        template<typename T > inline constexpr T phi
                        template<typename T > inline constexpr T catalan
                        template<typename T > inline constexpr T apery
                        template<typename T > inline constexpr T glaisher
            } //templates
        } //math_constants
}//std
```

The initialization part of these definitions will be implementation-specific. The implementation may at its discretion supply specializations of these variable templates for some or all fundamental floating-point types. The following requirements however need to be imposed:

1) Every implementation has to guarantee that a math constant could be instantiated if and only if its type is directly constructable from a fundamental floating-point type. For example, all types from the <complex> header would satisfy this requirement. Another example would be the following class:

```
struct sfloat : public std::string {
    sfloat(float f) :std::string(std::to_string(f)) {
    }
};
```

2) Every implementation needs to ensure that the instantiations of these template variables for fundamental floating-point types are the most accurate approximations of underlying real numbers for these types (design goal 2)). This entails that if two implementations provide fundamental floating-point types with identical lengths of significands, the constants instantiated for these types will be equal. For example, all IEEE-754 compliant implementations will have the value of pi<double> equal to 0x1.921fb54442d18p+1. This will guarantee that all C++ numerical libraries having the same internal precision will have identical values of their respective math constants.

After the templated constants, the following definitions should be made:

```
inline constexpr float ef = e_t<float>;
inline constexpr float log2ef = log2e_t<float>;
inline constexpr float log10ef = log10e_t<float>;
inline constexpr float pif = pi_t<float>;
inline constexpr float invpif = invpi_t<float>;
inline constexpr float invsqrtpif = invsqrtpi_t<float>;
inline constexpr float ln2f = ln2_t<float>;
inline constexpr float ln10f = ln10_t<float>;
```

```
inline constexpr float sqrt2f = sqrt2_t<float>;
inline constexpr float sqrt3f = sqrt3_t<float>;
inline constexpr float invsqrt3f = invsqrt3_t<float>;
inline constexpr float radianf = radian_t<float>;
inline constexpr float egammaf = egamma_t<float>;
inline constexpr float phif = phi_t<float>;
inline constexpr float catalanf = catalan_t<float>;
inline constexpr float aperyf = apery_t<float>;
inline constexpr float glaisherf = glaisher_t<float>;

inline constexpr double e = e_t<double>;
inline constexpr double log2e = log2e_t<double>;
inline constexpr double log10e = log10e_t<double>;
inline constexpr double pi = pi_t<double>;
inline constexpr double invpi = invpi_t<double>;
inline constexpr double invsqrtpi = invsqrtpi_t<double>;
inline constexpr double ln2 = ln2_t<double>;
inline constexpr double ln10 = ln10_t<double>;
inline constexpr double sqrt2 = sqrt2_t<double>;
inline constexpr double sqrt3 = sqrt3_t<double>;
inline constexpr double invsqrt3 = invsqrt3_t<double>;
inline constexpr double radian = radian_t<double>;
inline constexpr double egamma = egamma_t<double>;
inline constexpr double phi = phi_t<double>;
inline constexpr double catalan = catalan_t<double>;
inline constexpr double apery = apery_t<double>;
inline constexpr double glaisher = glaisher_t<double>;

inline constexpr long double el = e_t<long double>;
inline constexpr long double log2el = log2e_t<long double>;
inline constexpr long double log10el = log10e_t<long double>;
inline constexpr long double pil = pi_t<long double>;
inline constexpr long double invpil = invpi_t<long double>;
inline constexpr long double invsqrtpil = invsqrtpi_t<long double>;
inline constexpr long double ln2l = ln2_t<long double>;
inline constexpr long double ln10l = ln10_t<long double>;
inline constexpr long double sqrt2l = sqrt2_t<long double>;
inline constexpr long double sqrt3l = sqrt3_t<long double>;
inline constexpr long double invsqrt3l = invsqrt3_t<long double>;
inline constexpr long double radianl = radian_t<long double>;
inline constexpr long double egammal = egamma_t<long double>;
inline constexpr long double phil = phi_t<long double>;
inline constexpr long double catalanl = catalan_t<long double>;
inline constexpr long double aperyl = apery_t<long double>;
inline constexpr long double glaisherl = glaisher_t<long double>;

The way these variable and variable template definitions are injected into
std::math_constants will be implementation-specific.
```

4.4. How will they be accessed?

Because the standard won't provide a drop-in replacement for POSIX/OpenCL/GNU constants, it will be up to the user how, or even if, to transition to standardized constants. Some motivated users may do this via a global search-and-replace. It is likely however that many C++ projects will have the standard constants introduced alongside with the extant POSIX or user-defined constants. This may cause

readability problems as well as subtle computational issues. For example, let's consider the following code fragment:

```
#define _USE_MATH_DEFINES
#include "math.h"

template<typename T> constexpr T pi =3.14159265358979323846L;

constexpr long double MY_OLD_PI = M_PI; //has been here for 10+ years
constexpr long double MY_NEW_PI = pi<long double>;

static_assert(MY_OLD_PI == MY_NEW_PI, "OMG!");
```

It compiles on Windows, where long double is 64-bit, but fails on Linux, where it is 128-bit.

If an existing codebase already has user-defined math constants, the hard-coded floating-point literals in their definitions can easily be replaced with standard constants, for example:

```
const double PI = std::math_constants::pi;
```

In a more "greenfield" situation, where math constants are just being introduced, they can be imported into a global scope by the using directive, such as:

```
using std::math_constants::pi;
```

5. A "Hello world" program for math constants

```
#include <numeric>

using std::math_constants::pi;
using std::math_constants::pi_t;

template<typename T> constexpr T circle_area(T r) { return pi_t<T> * r * r; }

int main()
{
	static_assert(!!pi);
	static_assert(!!circle_area(1.0));
	return 0;
}
```

6. Proposed Changes in the Standard

§ 29.8.1 Header <numeric> synopsis

After

```
// 29.8.14, least common multiple
template <class M, class N>
	constexpr common_type_t<M, N> lcm(M m, N n);
```

the following should be inserted:

```
// 29.8.15 mathematical constants
namespace math_constants {
              template<typename T > inline constexpr T e_t           see below
              template<typename T > inline constexpr T log2e_t       see below
              template<typename T > inline constexpr T log10e_t      see below
              template<typename T > inline constexpr T pi_t          see below
              template<typename T > inline constexpr T invpi_t       see below
              template<typename T > inline constexpr T invsqrtpi_t   see below
              template<typename T > inline constexpr T ln2_t         see below
              template<typename T > inline constexpr T ln10_t        see below
              template<typename T > inline constexpr T sqrt2_t       see below
              template<typename T > inline constexpr T sqrt3_t       see below
              template<typename T > inline constexpr T invsqrt3_t    see below
              template<typename T > inline constexpr T radian_t      see below
              template<typename T > inline constexpr T egamma_t      see below
              template<typename T > inline constexpr T phi_t         see below
              template<typename T > inline constexpr T catalan_t     see below
              template<typename T > inline constexpr T apery_t       see below
              template<typename T > inline constexpr T glaisher_t    see below

              inline constexpr float ef = e_t<float>;
              inline constexpr float log2ef = log2e_t<float>;
              inline constexpr float log10ef = log10e_t<float>;
              inline constexpr float pif = pi_t<float>;
              inline constexpr float invpif = invpi_t<float>;
              inline constexpr float invsqrtpif = invsqrtpi_t<float>;
              inline constexpr float ln2f = ln2_t<float>;
              inline constexpr float ln10f = ln10_t<float>;
              inline constexpr float sqrt2f = sqrt2_t<float>;
              inline constexpr float sqrt3f = sqrt3_t<float>;
              inline constexpr float invsqrt3f = invsqrt3_t<float>;
              inline constexpr float radianf = radian_t<float>;
              inline constexpr float egammaf = egamma_t<float>;
              inline constexpr float phif = phi_t<float>;
              inline constexpr float catalanf = catalan_t<float>;
              inline constexpr float aperyf = apery_t<float>;
              inline constexpr float glaisherf = glaisher_t<float>;

              inline constexpr double e = e_t<double>;
              inline constexpr double log2e = log2e_t<double>;
              inline constexpr double log10e = log10e_t<double>;
              inline constexpr double pi = pi_t<double>;
              inline constexpr double invpi = invpi_t<double>;
              inline constexpr double invsqrtpi = invsqrtpi_t<double>;
              inline constexpr double ln2 = ln2_t<double>;
              inline constexpr double ln10 = ln10_t<double>;
              inline constexpr double sqrt2 = sqrt2_t<double>;
              inline constexpr double sqrt3 = sqrt3_t<double>;
              inline constexpr double invsqrt3 = invsqrt3_t<double>;
              inline constexpr double radian = radian_t<double>;
              inline constexpr double egamma = egamma_t<double>;
              inline constexpr double phi = phi_t<double>;
              inline constexpr double catalan = catalan_t<double>;
              inline constexpr double apery = apery_t<double>;
              inline constexpr double glaisher = glaisher_t<double>;
```

```
            inline constexpr long double el = e_t<long double>;
            inline constexpr long double log2el = log2e_t<long double>;
            inline constexpr long double log10el = log10e_t<long double>;
            inline constexpr long double pil = pi_t<long double>;
            inline constexpr long double invpil = invpi_t<long double>;
            inline constexpr long double invsqrtpil = invsqrtpi_t<long double>;
            inline constexpr long double ln2l = ln2_t<long double>;
            inline constexpr long double ln10l = ln10_t<long double>;
            inline constexpr long double sqrt2l = sqrt2_t<long double>;
            inline constexpr long double sqrt3l = sqrt3_t<long double>;
            inline constexpr long double invsqrt3l = invsqrt3_t<long double>;
            inline constexpr long double radianl = radian_t<long double>;
            inline constexpr long double egammal = egamma_t<long double>;
            inline constexpr long double phil = phi_t<long double>;
            inline constexpr long double catalanl = catalan_t<long double>;
            inline constexpr long double aperyl = apery_t<long double>;
            inline constexpr long double glaisherl = glaisher_t<long double>;
}
```

After **§29.8.14**, a new section **§29.8.15** should be inserted:

29.8.15 Mathematical constants

```
namespace math_constants {
            template<typename T > inline constexpr T e_t           see below
            template<typename T > inline constexpr T log2e_t       see below
            template<typename T > inline constexpr T log10e_t      see below
            template<typename T > inline constexpr T pi_t          see below
            template<typename T > inline constexpr T invpi_t       see below
            template<typename T > inline constexpr T invsqrtpi_t   see below
            template<typename T > inline constexpr T ln2_t         see below
            template<typename T > inline constexpr T ln10_t        see below
            template<typename T > inline constexpr T sqrt2_t       see below
            template<typename T > inline constexpr T sqrt3_t       see below
            template<typename T > inline constexpr T invsqrt3_t    see below
            template<typename T > inline constexpr T radian_t      see below
            template<typename T > inline constexpr T egamma_t      see below
            template<typename T > inline constexpr T phi_t         see below
            template<typename T > inline constexpr T catalan_t     see below
            template<typename T > inline constexpr T apery_t       see below
            template<typename T > inline constexpr T glaisher_t    see below

            inline constexpr float ef = e_t<float>;
            inline constexpr float log2ef = log2e_t<float>;
            inline constexpr float log10ef = log10e_t<float>;
            inline constexpr float pif = pi_t<float>;
            inline constexpr float invpif = invpi_t<float>;
            inline constexpr float invsqrtpif = invsqrtpi_t<float>;
            inline constexpr float ln2f = ln2_t<float>;
            inline constexpr float ln10f = ln10_t<float>;
            inline constexpr float sqrt2f = sqrt2_t<float>;
            inline constexpr float sqrt3f = sqrt3_t<float>;
            inline constexpr float invsqrt3f = invsqrt3_t<float>;
            inline constexpr float radianf = radian_t<float>;
            inline constexpr float egammaf = egamma_t<float>;
            inline constexpr float phif = phi_t<float>;
```

```
    inline constexpr float catalanf = catalan_t<float>;
    inline constexpr float aperyf = apery_t<float>;
    inline constexpr float glaisherf = glaisher_t<float>;

    inline constexpr double e = e_t<double>;
    inline constexpr double log2e = log2e_t<double>;
    inline constexpr double log10e = log10e_t<double>;
    inline constexpr double pi = pi_t<double>;
    inline constexpr double invpi = invpi_t<double>;
    inline constexpr double invsqrtpi = invsqrtpi_t<double>;
    inline constexpr double ln2 = ln2_t<double>;
    inline constexpr double ln10 = ln10_t<double>;
    inline constexpr double sqrt2 = sqrt2_t<double>;
    inline constexpr double sqrt3 = sqrt3_t<double>;
    inline constexpr double invsqrt3 = invsqrt3_t<double>;
    inline constexpr double radian = radian_t<double>;
    inline constexpr double egamma = egamma_t<double>;
    inline constexpr double phi = phi_t<double>;
    inline constexpr double catalan = catalan_t<double>;
    inline constexpr double apery = apery_t<double>;
    inline constexpr double glaisher = glaisher_t<double>;

    inline constexpr long double el = e_t<long double>;
    inline constexpr long double log2el = log2e_t<long double>;
    inline constexpr long double log10el = log10e_t<long double>;
    inline constexpr long double pil = pi_t<long double>;
    inline constexpr long double invpil = invpi_t<long double>;
    inline constexpr long double invsqrtpil = invsqrtpi_t<long double>;
    inline constexpr long double ln2l = ln2_t<long double>;
    inline constexpr long double ln10l = ln10_t<long double>;
    inline constexpr long double sqrt2l = sqrt2_t<long double>;
    inline constexpr long double sqrt3l = sqrt3_t<long double>;
    inline constexpr long double invsqrt3l = invsqrt3_t<long double>;
    inline constexpr long double radianl = radian_t<long double>;
    inline constexpr long double egammal = egamma_t<long double>;
    inline constexpr long double phil = phi_t<long double>;
    inline constexpr long double catalanl = catalan_t<long double>;
    inline constexpr long double aperyl = apery_t<long double>;
    inline constexpr long double glaisherl = glaisher_t<long double>;
}
```

[1] *Requires:* T shall be directly constructable from a fundamental floating-point type.

[2] *Remarks:* These variable templates should be initialized with implementation-defined values of e, $\log_2 e$, $\log_{10} e$, π, $\frac{1}{\pi}$, $\frac{1}{\sqrt{\pi}}$, ln2, ln10, $\sqrt{2}$, $\sqrt{3}$, $\frac{1}{\sqrt{3}}$, $\frac{180}{\pi}$, Euler-Mascheroni γ constant, golden ratio ϕ constant ($\frac{1+\sqrt{5}}{2}$), Catalan's constant, Apéry's $\zeta(3)$ constant and Glaisher's constant, respectively. The implementation may provide their specializations for some or all fundamental floating-point types (see **3.9.1)**. For each fundamental floating-point type, an instantiation of every variable template should be equal to the closest approximation of the underlying real number among the type's set of values.

7. References

The POSIX version of math.h is described at
http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/math.h.html.

The OpenCL mathematical constants are defined in a file opencl_math_constants, see
https://raw.githubusercontent.com/KhronosGroup/libclcxx/master/include/opencl_math_constants.

The GNU math extensions: https://gcc.gnu.org/onlinedocs/gcc-6.1.0/libstdc++/api/a01120_source.html

A list of built-in Wolfram constants is at
http://reference.wolfram.com/language/tutorial/MathematicalConstants.html.en.

8. Acknowledgments

The author would like to thank Edward Smith-Rowland for his review of the draft proposal, Vishal Oza, Daniel Krügler and Matthew Woehlke for their participation in the related thread at the std-proposals user group, and Walter E. Brown for volunteering to present the proposal and numerous helpful comments.